GD-Gibbs: A GPU-based Sampling Algorithm for Solving Distributed Constraint Optimization Problems

(Extended Abstract)

Ferdinando Fioretto, Federico Campeotto, Luca Da Rin Fioretto,
William Yeoh, Enrico Pontelli
Department of Computer Science
New Mexico State University
Las Cruces, NM 88003, USA
{ffiorett,fcampeot,ldarinfi,wyeoh,epontell}@cs.nmsu.edu

ABSTRACT

Researchers have recently introduced a promising new class of *Distributed Constraint Optimization Problem* (DCOP) algorithms that is based on sampling. This paradigm is very amenable to parallelization since sampling algorithms require a lot of samples to ensure convergence, and the sampling process can be designed to be executed in parallel. This paper presents *GPU-based D-Gibbs* (GD-Gibbs), which extends the Distributed Gibbs (D-Gibbs) sampling algorithm and harnesses the power of parallel computation of GPUs to solve DCOPs. Experimental results show that GD-Gibbs is faster than several other benchmark algorithms on a distributed meeting scheduling problem.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed AI—*Multiagent Systems*

Keywords

DCOP; Sampling; Gibbs; GPU

1. INTRODUCTION

Distributed Constraint Optimization Problems (DCOPs) are problems where agents need to coordinate their value assignments to maximize the sum of resulting constraint utilities [4, 7, 8]. Researchers have recently introduced a promising new class of approximation algorithms that is based on sampling. The two current state-of-the-art DCOP sampling algorithms are DUCT [6] and D-Gibbs [5]. These algorithms have been shown to outperform existing local search algorithms like DSA and MGM in terms of convergence rate and the quality of the converged solution. The sampling paradigm is very amenable to parallelization since sampling algorithms require a lot of samples to ensure convergence, and the sampling process can be designed to be executed in parallel.

In this paper, we explore the use of general-purpose *Graphical Processing Units* (GPUs), which are powerful parallel architectures that are readily available in the form of graphic cards in most modern computers, to parallelize the sampling process of D-Gibbs. Specifically, we extend the D-Gibbs [5] and DPOP [7] algorithms to *GPU-based D-Gibbs* (GD-Gibbs), which harnesses the power of parallel computation of GPUs to solve DCOPs. It emulates the computation and communication operations of DPOP via the use of GPUs to perform Gibbs sampling. Our experimental results on distributed meeting scheduling problems show that GD-Gibbs is able to find better solutions up to two orders of magnitude faster than MGM and MGM2 (two local search DCOP algorithms).

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

2. BACKGROUND

DCOP: A *Distributed Constraint Optimization Problem* (DCOP) [4, 7, 8] is defined by $\langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \alpha \rangle$, where $\mathcal{X} = \{x_1, \dots, x_n\}$ is a set of *variables*; $\mathcal{D} = \{D_1, \dots, D_n\}$ is a set of finite *domains*, where D_i is the domain of variable x_i; $\mathcal{F} = \{f_1, \dots, f_m\}$ is a set of *utility functions* (also called *constraints*), where each k-ary utility function $f_i : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \mapsto \mathbb{N} \cup \{-\infty, 0\}$ specifies the utility of each combination of values of variables in its *scope* (i.e., $x_{i_1}, \dots, x_{i_k}$); $\mathcal{A} = \{a_1, \dots, a_p\}$ is a set of *agents*; and $\alpha : \mathcal{X} \to \mathcal{A}$ maps each variable to one agent. We use the notation $\mathcal{X}_i$ to denote the set of variables mapped to agent a_i. A *solution* is a value assignment for all variables. Its utility is the evaluation of all utility functions on that solution. The goal is to find a utility-maximal solution.

Gibbs and Distributed Gibbs: The *Gibbs* sampling algorithm [1] is a Markov chain Monte Carlo algorithm that can be used to approximate joint probability distributions. It generates a Markov chain of samples, each of which is correlated with previous samples. While the Gibbs algorithm is designed to solve the *maximum a posteriori* (MAP) estimation problem, it can also be used to solve DCOPs in a *centralized manner* by mapping MAP estimation problems to DCOPs [5]. The *Distributed Gibbs* (D-Gibbs) algorithm extends Gibbs by tailoring it to solve DCOPs in a *decentralized manner* [5]. The main difference between the two algorithms is that D-Gibbs exploits conditional independent subproblems by sampling the subproblems in parallel.

DPOP: The *Distributed Pseudo-tree Optimization Procedure* (DPOP) [7] is a complete DCOP algorithm and has the following three phases:

- **Pseudo-tree Generation Phase:** DPOP calls exist-

$\lvert\mathcal{X}_i\rvert$	10			25			50		
	wct	*st*	*quality*	*wct*	*st*	*quality*	*wct*	*st*	*quality*
DPOP	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
MGM	3910	136	2766	19250	673	11652	**8350**	20716	36373
MGM2	9260	756	2783	59120	4384	**11889**	278240	20971	36491
GD-Gibbs	**284**	**30**	**3173**	**3183**	**368**	11884	34210	**3439**	**42124**

$\lvert D_i\rvert$	12			24			48		
	wct	*st*	*quality*	*wct*	*st*	*quality*	*wct*	*st*	*quality*
DPOP	22230	9996	**1332**	timeout	timeout	-	timeout	timeout	-
MGM	3810	113	1085	4050	145	2690	4720	276	6319
MGM2	6090	279	1134	9800	761	2645	23810	2549	6660
GD-Gibbs	**214**	**22**	1297	**283**	**30**	**3186**	**363**	**41**	**7068**

Table 1: Experimental Results

ing distributed pseudo-tree construction algorithms to construct its pseudo-tree.

- **UTIL Propagation Phase:** Each agent, starting from the leafs of the pseudo-tree, computes the optimal sum of utilities in its subtree for each value combination of its ancestor agents. The agent does so by summing the utilities in the UTIL messages received from its child agents and then projecting out its own variables by optimizing over them.
- **VALUE Propagation Phase:** Each agent, starting from the root of the pseudo-tree, determines the optimal value for its variables. The root agent does so using the UTIL messages received in the second phase.

3. GPU-BASED DISTRIBUTED GIBBS

We now describe our *GPU-based Distributed Gibbs* (GD-Gibbs) algorithm, which extends D-Gibbs and DPOP. At a high level, its operations are similar to the operations of DPOP except that the computation of the utility tables sent by agents during the UTIL phase is done by sampling with GPUs. The computation of each row in a utility table is independent of the computation in the other rows, and GD-Gibbs exploits this independence and samples the utility in each row in parallel.

Like DPOP, there are also three phases in the operation of GD-Gibbs. The first phase is identical to that of DPOP. The second phase is similar to that of DPOP except that each agent computes the best utility and the corresponding solution for each row in the utility table *in parallel*. For each row, the agent computes the best utility by executing the Gibbs sampling procedure and takes multiple samples *in parallel*. By the end of the second phase, the root agent knows the overall utility for each combination of values of its variables. It chooses its best value combination that results in the maximum utility, and starts the third phase, which is identical to that of DPOP.

4. RESULTS AND CONCLUSIONS

We compare GD-Gibbs against DPOP [7] (an optimal algorithm) and MGM [3] and MGM2 [3] (sub-optimal algorithms). To compare runtimes and solution qualities, we use publicly-available implementations of MGM, MGM2, and DPOP, which are all implemented on the FRODO framework [2]. We run our experiments on a machine with two Intel(R) Xeon(R) X5650 with 2.67GHz CPU and 16GB of RAM; and 8 Tesla M2075 GPUs, each with 14 multiprocessors, 448 cores, and a clock rate of 1.15GHz. We measure runtime using the wall clock time (*wct*) and the simulated time (*st*) metrics and evaluate the algorithms on distributed meeting scheduling problems.

Table 1 show the results, where we vary only one parameter and fix the rest to their "default" values: $\lvert\mathcal{A}\rvert = 10, \lvert\mathcal{X}_i\rvert = 10, \lvert D_i\rvert = 24$. We set the number of samples for the GD-Gibbs algorithms to 100. The results show that GD-Gibbs can find reasonably good solutions (within 5% error) up to two orders of magnitude faster than MGM and MGM2.

In this paper, we take the first step towards harnessing the power of parallel computation of GPUs to solve DCOPs. We introduce the GD-Gibbs algorithm, which decomposes a DCOP into independent subproblems that can each be sampled in parallel by GPUs. Our preliminary results demonstrate the potential for using GPUs to scale up DCOP algorithms, which is exciting as GPUs provide access to hundreds of computing cores at a very affordable cost.

5. REFERENCES

[1] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[2] T. Léauté, B. Ottens, and R. Szymanek. FRODO 2.0: An open-source framework for distributed constraint optimization. In *Proc. of the Distributed Constraint Reasoning Workshop*, pages 160–164, 2009.

[3] R. Maheswaran, J. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical game-based approach. In *Proc. of PDCS*, pages 432–439, 2004.

[4] P. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1–2):149–180, 2005.

[5] D. T. Nguyen, W. Yeoh, and H. C. Lau. Distributed Gibbs: A memory-bounded sampling-based DCOP algorithm. In *Proc. of AAMAS*, pages 167–174, 2013.

[6] B. Ottens, C. Dimitrakakis, and B. Faltings. DUCT: An upper confidence bound approach to distributed constraint optimization problems. In *Proc. of AAAI*, pages 528–534, 2012.

[7] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proc. of IJCAI*, pages 1413–1420, 2005.

[8] W. Yeoh and M. Yokoo. Distributed problem solving. *AI Magazine*, 33(3):53–65, 2012.